REMARKS

This communication is in response to the Office Action mailed June 27, 2005. The Examiner now rejects the amended claims as being obvious over Schmidt (previously relied upon) in combination with Toutonghi.

The independent claims have been further amended and, as discussed below, it is respectfully submitted that the combination of Schmidt and Toutonghi does not render unpatentable the subject matter recited therein. In addition, the claims are amended as appropriate to address the matters under 35 U.S.C. § 101.

Statutory Subject Matter

Claims 1-3 are amended to recite a "computer-implemented" method as suggested by the Examiner. In addition, claims 4-9 have been amended to more clearly recite statutory subject matter. It is respectfully requested that the rejection under 35 U.S.C. § 101 be withdrawn.

If the Examiner continues to reject claims 4-9 as being directed to non-statutory subject matter, Applicant would appreciate suggested language that would be acceptable to the Examiner.

Summary of Subject Matter Recited in Claims

Without intending to limit or otherwise affect the scope of the claims, it is useful to review the subject matter that is claimed.

Claim 1 recites a method for requesting a consistent state so that, for example, garbage collection or other global safe-point operations may be safely performed. As discussed, for example, at page 7, lines 20-24:

> The present invention enables a determination to be made as to which threads in a system are inconsistent, or "garbage collection unsafe," and cause them to reach a consistent, or "garbage collection safe," point without requiring all threads to be suspended. In one embodiment, inconsistent threads may be caused to reach a consistent point with very few synchronization operations.

As further discussed at page 8, lines 1-5:

> As allowing inconsistent threads to reach a consistent state generally occurs much more frequently that allowing a requesting thread to request a consistent state, the use of consistent and inconsistent states in accordance with the present invention increases the overall efficiency and performance of a multi-threaded computing environment.

Finally, as further discussed at page 9, lines 9-17:

5

A rendezvous operation is generally arranged to enable the thread to determine its status with respect to a requestor thread, e.g., the thread which requested a check point. A rendezvous operation also synchronizes between a reader thread and a writer thread, and handles notification of a writer and blocking of the reader, if necessary.

Claim 1 has been amended to further specify that the "saving a snapshot of <u>an indication of</u> a state of the first thread and thereafter setting the state of the first thread to a safe state, <u>wherein the indication of the state of which the snapshot is saved is an indication of whether or not the first thread was consistent</u>." See, for example, page 13, line 28 et seq., which describes this feature. As discussed at page 11, line 29 et seq.:

> After the consistent state "requested" flag is set to true in step 410, the requesting thread begins to iterate over all threads in a multi-threaded environment except for itself in step 414. For each thread "i" which is not the requesting thread, a "was consistent" flag is set to indicate the current state for each thread "i" in step 418.

The "was consistent" state indication may be used, for example, in a process (such as shown in Fig. 4) to determine the consistency of substantially all threads in a multithreaded environment. Specifically, the indication for a particular thread may be used to determine if that thread is consistent. By saving and subsequently restoring the "was consistent" state indication, the rendezvous operation does not change this state indication.

The other independent claims are similar in substance to claim 1.

The Prior Art-Based Rejection

The claims are rejected based on a combination of Schmidt and Toutonghi. The Schmidt disclosure is similar to that discussed in the background of Applicant's specification. Without taking a position at this time, Applicant accepts (for the purposes of the present response only, and with reservation of the option to later challenge) the Examiner's position that Schmidt discloses what is recited in claim 1 except for the "saving" and "restoring" steps.

It is respectfully submitted that Toutonghi fails to satisfy the "saving" and "restoring" steps. In particular, the cited portion of Toutonghi discloses saving and restoring the general contents of a thread's registers upon the entrance and exit, respectively, of the SuspendReturn procedure. This saving and restoring appears to be necessary because the SuspendReturn routine in which the saving and restoring is called as part of a "hijack" scheme. That is, as disclosed for example at col. 12, lines 59-60, "the SuspendReturn program is executed when the call that was being made when the thread was last hijacked returns." The "hijack" scheme is essentially a patch (change of control temporarily outside the original path of execution) to the normal program code.

The use of the "hijack" scheme is discussed more generally at col. 10, lines 43-55:

6

A second technique involves patching, or "hijacking," returns from a calling program to a called program in the StartGarbageCollection function where the thread executing the called program has disabled garbage collection, and where the called program is not interruptible. The hijacking technique is based upon the recognition that a call return is a safe time to interrupt the execution of a thread. As a result of this hijacking, when the called program returns, it returns not to the calling program, but to a SuspendReturn program that causes the thread to be suspended by calling the EnableGarbageCollection API, then calling the DisableGarbageCollection API. At the end of the SuspendReturn program, the thread returns to the calling program.

It appears that, since the SuspendReturn program is a program not intended (at least, by the programmers) to have been called the SuspendReturn program saves, and then restores, the registers that it uses.

The "state indication" of which a snapshot is saved is not merely, in general, the contents of any of the thread's registers that are used by a program. In general, when a program (e.g., an interrupt routine, that is invoked asynchronously) saves and then restores registers, the determination of what registers to save and restore is on the hardware level (i.e., what hardware registers are being used by the interrupting program) and does not consider at all what the data stored in the hardware registers represents.

For at least the reasons discussed, then, the combination of Schmidt and Toutonghi does not yield the subject matter of claim 1. Furthermore, there is nothing in the cited references to suggest modifying Schmidt, Toutonghi, or the combination thereof, to save and restore the "state indication" as recited in claim 1.

The other independent claims have been similarly amended. With respect to the rejection of the other independent claims, then, Applicant incorporates herein the comments made above relative to the rejection of claim 1.

7

CONCLUSION

Applicants believe that all pending claims are allowable and respectfully requests a Notice of Allowance for this application from the Examiner. Should the Examiner believe that a telephone conference would expedite the prosecution of this application, the undersigned can be reached at the telephone number set out below.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

Alan S. Hodes
Reg. No. 38,185

P.O. Box 70250
Oakland, CA 94612-0250
(650) 961-8300

8